МИНОБРНАУКИ РОССИИ

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ

ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

Кафедра МО ЭВМ

ОТЧЕТ по лабораторной работе №2 по дисциплине «Алгоритмы и структуры данных» Тема: Рекурсивная обработка иерархических списков Вариант 11

Студент гр. 8304 Масалыкин Д.Р.

Преподаватель Фирсов М.А.

Санкт-Петербург

2019

Цель работы.

Познакомиться с нелинейной конструкцией – иерархический список, способами её организации и рекурсивной обработки. Получить навыки решения задач обработки иерархических списков, с использованием базовых функций их рекурсивной обработки.

Постановка задачи.

1) проанализировать полученное задание, выделив рекурсивно определяемые информационные объекты и (или) действия;
2) разработать программу, для решения поставленного задания, использующую рекурсию;
3) сопоставить рекурсивное решение с итеративным решением задачи;
4) сделать вывод о целесообразности и эффективности рекурсивного решения данной задачи.

Сформировать линейный список атомов исходного иерархического списка таким образом, что скобочная запись полученного линейного списка будет совпадать с сокращённой скобочной записью исходного иерархического списка после устранения всех внутренних скобок;

Описание алгоритма.

Программа рекурсивно считывает данные, проверяет их корректность и заносит их в список. Для этого считывается очередной символ строки. Если это атом, он добавляется к иерархическому списку, если это список – рекурсивно заносится в список.

Для создания линейного списка программа проходит по иерархическому списку и спускается на уровень ниже если встречает такой переход. После прохода по нижнему уровню программа возвращается к прежней позиции.

создание_линейного_списка(список_и, список_л)

{ если атом

вставка_назад

иначе

переход на нижний уровень

повторять_пока(хвост_не_пуст)

Спецификация программы.

Программа предназначена для создания линейного списка на основе иерархического.

Программа написана на языке C++ с использованием фреймворка Qt. Входными данными являются символы английского алфавита и скобки - считываются из полей lineEdit. Выходными данными являются промежуточные значения вычисления выражения и конечный результат. Данные выводятся в qDebug(), результат показывается во всплывающем окне.

Тестирование.

```
Создание линейного списка:
Проверка на корректность: (A(B(C)D)E)
Строка корректна.
Строка корректна, иерархический список создан
__Считанный спискок__:
Список: ( A ( B ( C ) D ) E )
(ABCDE)

Создание линейного списка:
Проверка на корректность: (A(B(C)D)E)
Строка корректна.
Строка корректна, иерархический список создан
__Считанный спискок__:
Список: ( A ( B ( C ) D ) E )
(ABCDE)
```

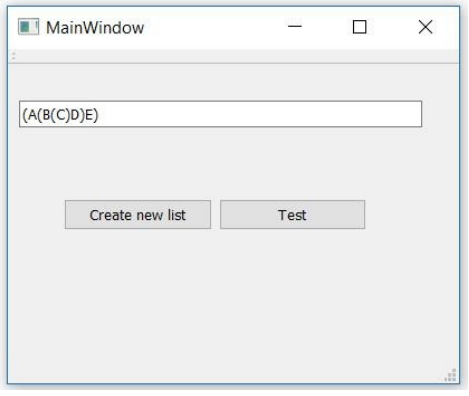


Рисунок 1- сравнение списков

Остальные тесты приведены в табл. 1.

Таблица 1 – Результаты тестирования программы

Ввод	Результат
qwe()(ad))	Некорректные данные
(	Некорректные данные
(qwer(sadas)(asfas)sda()	Некорректные данные
((qwer()(vddkbk))(ervnvn()ekvnnlkewnvkl(ifuhiuvhi)evenn)wevnjvndn(()()edv)) ((vde()())ndnvjnvew(nneve(ihvuihufi)lkvnweklnnvke()nvnvre)((kbkddv)()rewq))	Список создан
(weq(weq((weq)(weq)(we))egdb)egdb) (bdge(bdge((ew)(qew)(qew))qew)qew)	Список создан
(qwer)	Список создан
() ()	Список создан
((((())))) ((((()))))	Список создан

Анализ алгоритма.

Алгоритм работает за линейное время от размера списка. Недостаток рекурсивного алгоритма – ограниченный стек вызовов функций, что в свою очередь накладывает ограничение на количество вложенных списков, а также затраты производительности на вызов функций.

Описание функций и СД.

Класс-реализация иерархического списка содержит умный указатель на вложенный список (Head), умный указатель на следующий элемент списка (Tail), флаг, для определения атома, значение атома. Умные указатели были использованы во избежание утечек памяти.

Методы класса для доступа к данным.

```
ListPointer getHead() const;
ListPointer getTail() const;
bool isNull() const;
bool isAtom() const;
    char getAtom() const;
```

Статический метод класса для создания иерархического списка:

```
static bool buildList(HierarchicalList::ListPointer& list, const std::string& str);
```

Принимает на вход ссылку на строку и ссылку на умный указатель на список, проверяет корректность строки и вызывает функции рекурсивного создания списка.

Метод класса для рекурсивной печати следующих списков.

```
void print_seq(std::ostream& out) const;
```

Принимает на вход ссылку на выхоной поток и рекурсивно выводит следующие списки.

Приватный метод класса для считывания вложенных списков:

```
static void readData(ListPointer& list, const char prev,
std::string::const_iterator& it,
        const std::string::const_iterator& end);
```

Принимает на вход ссылку на умный указатель на список, предыдущий элемент строки и итераторы на строку. Если предыдущий элемент не равен “(“, создается атом, в ином случае вызывает считывание следующего списка.

Приватный метод класса для считывания следующих списков:

```
static void readSeq(ListPointer& list, std::string::const_iterator& it,
    const std::string::const_iterator& end);
```

Принимает на вход ссылку на умный указатель на список, предыдущий элемент строки и итераторы на строку. Если строка пустая – происходит return. Если элемент равен “)”, создается пустой список. В ином случае рекурсивно считываются вложенные и следующие списки, затем объединяются в текущем списке.

Статический метод класса для создания нового линейного списка:

```
void create_new_list(MyList::MyListP inp_list, std::list<char>* new_list, int depth);
```

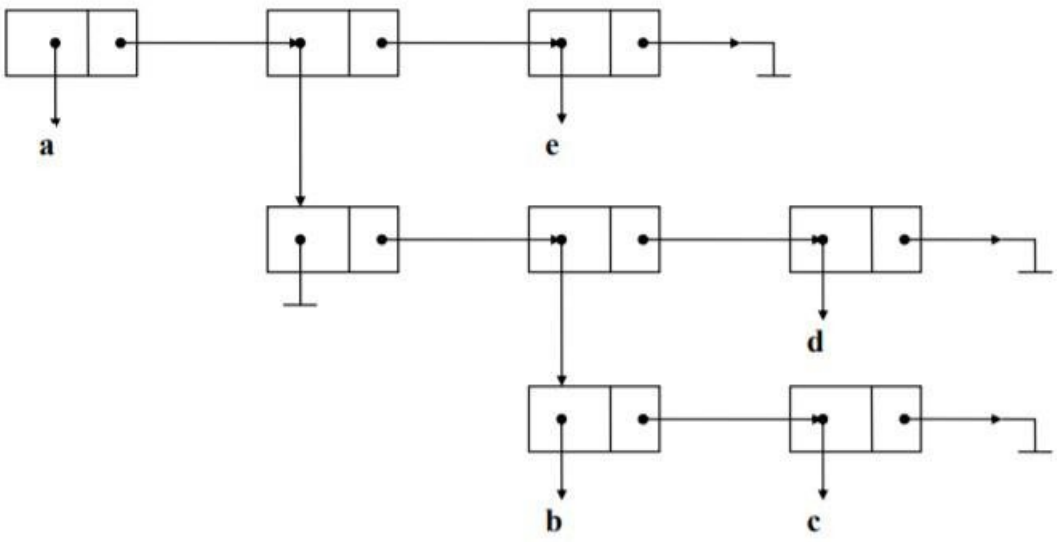
Принимает на вход ссылку на указатель на список, указатель на линейный список и глубину рекурсии. Для вложенного списка вызывается рекурсия, затем создается линейный список.

Функция для проверки корректности строки.

```
bool isCorrectStr(const std::string& str);
```

Принимает на вход ссылку на строку, проверяет размер и структуру строки, возвращает true, если строка корректна, и false в ином случае

Рисунок 2 – Структура иерархического списка Структура списка (a(()(bc)d)e)



Вывод.

В ходе выполнения данной лабораторной работы реализован класс иерархического списка и функция для реверсирования заданного писка. Научилась обрабатывать иерархические списки, с использованием базовых функций их рекурсивной обработки и сравнивать. Реализовано unit-тестирование.

Приложение А. Исходный код программы.

MyList.h

```
//
// Created by Dan on 20.10.2019.
//

#ifndef ADS_LR2_MYLIST_H
#define ADS_LR2_MYLIST_H



#include <vector>
#include <string>
#include <iostream>
#include <memory>
#include <list>

class MyList{

    /*
     * РљР»Р°СЃСЃ РґР»СЏ СЂР°Р±РѕС‚С‹ СЃ РёРµСЂР°СЂС…РёС‡РµСЃРєРёРјРё СЃРїРёСЃРєР°РјРё
    */
public:
    typedef std::shared_ptr<MyList> MyListP;

    explicit MyList();
    ~MyList();

    MyList& operator=(const MyList& list) = delete;
    MyList(const MyList& list) = delete;

    MyListP getHead() const;
    MyListP getTail() const;
    bool isNull() const;
    bool getIsAtom() const;
    char getAtom() const;

    friend std::ostream& operator<< (std::ostream& out, const MyListP list);
    static bool buildList(MyListP& list, const std::string& str);

private:
    static bool checkStr(const std::string& str);
    static void readData(MyListP& list, const char prev, std::string::const_iterator& it,
                         const std::string::const_iterator& end);
    static void readSeq(MyListP& list, std::string::const_iterator& it,
                        const std::string::const_iterator& end);
    static MyListP cons(MyListP& head, MyListP& tail);
    void createAtom(const char ch);
    void print_seq(std::ostream& out) const;

private:
    bool isAtom;
    MyListP head;
    MyListP tail;
    char atom;
};

#endif //ADS_LR2_MYLIST_H
```

main.cpp

```
#include <iostream>
#include <fstream>
#include "MyList.h"

void create_new_list(MyList::MyListP inp_list, std::list<char>* new_list, int depth);
void print_list(std::list<char> list);

int main(int argc, char* argv[]) {
    std::string inpStr;
    if(argc == 1){
```

```
     std::cout<<"Enter a hierarchical list: ";
     std::cin>>inpStr;
    }
    else{
     std::string fileName = argv[1];
        //result += "Read text from file " + fileName + "\n";

        std::ifstream inputFile(fileName, std::ios::in);
        //inpStr = readText(inputFile);
     getline(inputFile, inpStr);
        inputFile.close();
     }

    //создание списка, проверка на корректность
    MyList::MyListP List(new MyList);

    if (MyList::buildList(List, inpStr)) {
        std::cout << "The line is correct, the hierarchical list is created"<<std::endl;
    }
    else {
        std::cout<< "The string is incorrect!"<<std::endl;
        std::cerr<<"The input is not correct."<<std::endl;
        return 0;
    }

    std::cout<< "__ Read list__:"<<std::endl;
    std::cout << "List: " << List<<std::endl;
    std::list<char> new_list;
    new_list = {};
    create_new_list(List, &new_list, 0);
    std::cout<<"New list: ";
    print_list(new_list);
    return 0;
}

void create_new_list(MyList::MyListP inp_list, std::list<char>* new_list, int depth){
    depth++;
    if(inp_list->isNull())
        return;
    while(!inp_list->isNull()){
        if(inp_list->getHead()->getIsAtom()){
            new_list->push_back(inp_list->getHead()->getAtom());
            inp_list = inp_list->getTail();
        }
        else {
            create_new_list(inp_list->getHead(), new_list, depth);
            inp_list = inp_list->getTail();
        }
    }
}

void print_list(std::list<char> list){
    /*auto begin = list.begin();
    auto end = list.end();*/
    std::cout<<"(";
    for(auto i = list.begin(); i != list.end(); i++){
        std::cout <<*i;
    }
    std::cout <<")"<<std::endl;
}
```

mylist.cpp

```
//
// Created by Dan on 20.10.2019.
//

#include "MyList.h"

MyList::MyList()
{
    /*
     * РџРs СґРjРsР»С‡Р°РSРёСћ РsР±СЉРµРєС, РєР»Р°СЃСґР° СЏРIР»СЏРµС,СЃСЏ РїСґСЃС,С‹Pj
СЃРїРёСЃРєРsPj
    */
```

```
    isAtom = false;
    atom = 0;
    head = nullptr;
    tail = nullptr;
}


MyList::~MyList()
{
    /*
     * Рў.Рє PI РєР»Р°СЃСЃРµ РёСЃРїРѕР»СЊР·СѓСЋ СѓРјРЅС‹Рµ
РѕСЃРІРѕР¶РґРµРЅРёРµ
     * РїР°РјСЏС‚Рё РїСЂРѕРёСЃ…РѕРёС, Р°РІС,РѕРјР°С,РёС‡РµСЃРєРё
     */
}


bool MyList::isNull() const
{
    /*
     * Р’РѕР·РІСЂР°С‰Р°РµС, true, РµСЃР»Рё СЌР»РµРјРµРЅС, СЏРІР»СЏРµС,СЃСЏ РЅСѓР»РµРІС‹Рј
СЃРїРёСЃРєРѕРј,
     * false - PI РёРЅРѕРј СЃР»СѓС‡Р°Рµ
     */

    return (!isAtom && head == nullptr && tail == nullptr);
}


MyList::MyListP MyList::getHead() const
{
    /*
     * Р•СЃР»Рё СЌР»РµРјРµРЅС, РЅРµ Р°С,РѕРј - РІРѕР·Р±СЂР°С‰Р°РµС, СѓСЃР°
РІРѕРѕР¶РµРЅРёРµ СЃР»СѓС‡Р°Рµ,
     * PI РёРЅРѕРј СЃР»СѓС‡Р°Рµ - nullptr
     */

    if (!isAtom) {
        return head;
    }
    else {
        std::cerr << "Error: Head(atom)\n";
        return nullptr;
    }
}


MyList::MyListP MyList::getTail() const
{
    /*
     * Р•СЃР»Рё СЌР»РµРјРµРЅС, РЅРµ Р°С,РѕРј - РІРѕР·Р±СЂР°С‰Р°РµС, СѓСЃР°
СЃР»РµРґСѓСЋС‰РёР№ СЃРїРёСЃРѕРє,
     * PI РёРЅРѕРј СЃР»СѓС‡Р°Рµ - nullptr
     */

    if (!isAtom) {
        return tail;
    }
    else {
        std::cerr << "Error: Tail(atom)\n";
        return nullptr;
    }
}


bool MyList::getIsAtom() const
{
    /*
     * Р•СЃР»Рё СЌР»РµРјРµРЅС, Р°С,РѕРј - РІРѕР·Р±СЂР°С‰Р°РµС, true,
     * PI РёРЅРѕРј СЃР»СѓС‡Р°Рµ - false
     */

    return isAtom;
}


MyList::MyListP MyList::cons(MyListP& head, MyListP& tail)
{
    /*
```

```
     * РєСѓРЅРєС†РёСЏ СЃРѕР·РґР°РЅРёСЏ СЃРїРёСЃРєР°
    */
    if (tail != nullptr && tail->getIsAtom()) {
        std::cerr << "Error: Tail(atom)\n";
        return nullptr;
    }
    else {
        MyListP tmp(new MyList);
        tmp->head = head;
        tmp->tail = tail;
        return tmp;
    }
}




bool MyList::checkStr(const std::string& str)
{
    /*
     * РєСѓРЅРєС†РёСЏ РїСЂРѕРІРµСЂСЏРµС‚,РЅРѕСЂРјРµСЃ,Рё PIC…РѕРґРЅРѕ‹С… РґР°РЅРЅС‹С…,
     * РїСЂРёРЅРёРјР°РµС, РЅР° PIC…РѕРґ СЃСЃС‹Р»РєСѓ РЅР° СЃС,СЂСЃРєСѓ СЃС,СЂРѕРёРё,
РїСЂРѕРІРµСЂРєРµС, СЂР°Р·РјРµСЂ Рё СЃС,СЂСѓРєС СЃС,СЂРѕРёРё,
     * РІРѕР·РІСЂР°С‰Р°РµС, true, РµСЃР»Рё СЃС,СЂРѕРєР° РєРѕСЂСЂРµРєС,РЅР°, Рё false РІ РёРЅРѕРј
СЃР»СѓС‡Р°Рµ
    */

    std::cout << "Validation check:" << str.c_str()<<std::endl;
    int countBracket = 0;

    if (str.size() < 2)
        return false;

    if (str[0] != '(' || str[str.size() - 1] != ')')
        return false;

    size_t i;
    for (i = 0; i < str.size(); ++i) {
        char elem = str[i];
        if (elem == '('
            ++countBracket;
        else if (elem == ')')
            --countBracket;
        else if (!isalpha(elem))
            return false;

        if (countBracket <= 0 && i != str.size()-1)
            return false;
    }

    if (countBracket > 0 || i != str.size()) {
        return false;
    }

    std::cout<< "The line is correct."<<std::endl;
    return true;
}


void MyList::createAtom(const char ch)
{
    /*
     * РЎРѕР·РґР°РµС,СЃСЏ РѕР±СЉРµРєС, РєР»Р°СЃСЃР° - Р°С,РѕРј
    */
    this->atom = ch;
    this->isAtom = true;
}


void MyList::readData(MyListP &list, const char prev, std::string::const_iterator &it,
                      const std::string::const_iterator& end)
{
    /*
     * РєСѓРЅРєС†РёСЏ СЃС‡РёС,С‹РІР°РµС‚ РґР°РЅРЅС‹С…. РЎС‡РёС,С‹РІР°РµС, Р»РёР±Рѕ Р°С,РѕРј,
Р»РёРѕ СЂРµСѓСЂСЃРёРІРЅРѕ СЃС‡РёС,С‹РІР°РµС, СЃРїРёСЃРєР°
    */
```

```
    if (prev != '(') {
        list->createAtom(prev);
    }
    else {
        readSeq(list, it, end);
    }
}


void MyList::readSeq(MyListP& list, std::string::const_iterator&it,
                     const std::string::const_iterator& end)
{
    /*
     * P¤CґPSPєC†PëCЏ CЃC‡PëC,C‹PIP°PSPëCЏ CЃPїPëCЃPєP°. P PµPєCґCЃCЂPëPIPSPs
CЃC‡PëC,C‹PIP°PµC, PґP°PSPSC‹Pµ Pë CЃPїPëCЃPєPsPє Pë
     * PґPsP±P°PIP»CЏPµC, PëC… PI PëCЃC…PsPґPSC‹P№.
    */

    MyListP headList(new MyList);
    MyListP tailList(new MyList);

    if (it == end)
        return;

    if (*it == ')') {
        ++it;
    }
    else {
        char prev = *it;
        ++it;
        readData(headList, prev, it, end);
        readSeq(tailList, it, end);
        list = cons(headList, tailList);
    }
}


bool MyList::buildList(MyListP& list, const std::string& str)
{
    /*
     * P¤CґPSPєC†PëCЏ CЃPsP·PґP°PSPëCЏ PëPµCЂP°CЂC…PëC‡PµCЃPєPsPiPs CЃPїPëCЃPєP°. PµCЂPëPSPëPjP°PµC,
PSP° PIC…PsPґ CЃCЃC‹P»PєCґ
     * PSP° CЃC,CЂPsPєCґ, PїCЂPsPIPµCЂCЏPµC, PєPsCЂCЂPµPєC,PSPsCЃC,CЊ CЃC,CЂPsPєPë Pë
PIC‹P·C‹PIP°PµC, PїCЂPëPIP°C,PSC‹P№ PjPµC,PsPґ
     * readData().
    */


    if (!checkStr(str))
        return false;

    auto it_begin = str.cbegin();
    auto it_end = str.cend();
    char prev = *it_begin;
    ++it_begin;
    readData(list, prev, it_begin, it_end);

    return true;
}


char MyList::getAtom() const
{
    /*
     * P¤CґPSPєC†PëCЏ PIPsP·PICЂP°C‰P°PµC, P·PSP°C‡PµPSPëPµ P°C,PsPjP°
    */
    if (isAtom) {
        return atom;
    }
    else {
        std::cerr << "Error: getAtom(!atom)\n";
        return 0;
    }
}

std::ostream& operator<<(std::ostream& out, const MyList::MyListP list)
{
    /*
     * PµPµCЂPµPiCЂCґP·PєP° PsPїPµCЂP°C,PsCЂP° PIC‹PIPsPґP° PґP»CЏ PsC,P»P°PґPєPë PїPsPiCЂP°PjPjC‹
```

```
        */

        if (list == nullptr || list->isNull()) {
            out << "()";
        }
        else if (list->getIsAtom()) {
            out << list->getAtom();
        }
        else {
            out << "(";

            out << list->getHead();
            if (list->getTail() != nullptr)
                list->getTail()->print_seq(out);

            out << ")";
        }

        return out;
    }

    void MyList::print_seq(std::ostream& out) const
    {
        /*
         * P¤CґPSPєC†PëCЏ PїPµC‡P°C,Pë Tail
        */

        if (!isNull()) {
            out << this->getHead();

            if (this->getTail() != nullptr)
                this->getTail()->print_seq(out);
        }
    }
```

Makefile

```
#!/bin/bash

make


for file in ./Tests/*

do


./lab2 $file


done
```